コンポーネント再利用による
システム開発の工業化

なかむら　のぶひろ

中村　　伸裕

［略　歴］
1988 年　住友電気工業株式会社入社
現在　　情報システム部　システム技術課　主査
システム経験年数　13 年

たにもと　おさむ

谷本　　収

［略　歴］
1986 年　住友電気工業株式会社入社
現在　　住友電工情報システム株式会社
　　ビジネスソリューション開発部　部長補佐
システム経験年数　15 年

原稿量
本文　　10,000 字
要約　　1,189 字
図表　　12 枚

＜要　　約＞

近年、ソフトウエアの部品化がやりやすいオブジェクト指向言語 Java の登場により、コンポーネントを利用したシステム開発が注目されている。従来の労働集約型の開発から再利用可能なコンポーネントを組み合わせる組立型の開発形態に移行することで飛躍的な生産性の向上、コスト削減、納期短縮、品質向上を目指している。

我々は、コード体系に基づきデータ構造を分析するデータ中心設計を採用しており、事業部に最適なデータベースの上で再利用可能なコンポーネントを組み合わせてシステムを構築することを目指すことにした。1999 年に Java フレームワークを開発し、ソフトウエアの部品化と再利用の基盤を構築した。一般的に再利用できるのはビジネスロジック部分（モデル）と考えられているが、これまでの経験からモデルの再利用は難しいと判断したため、画面やデータベースの入出力から部品化に着手し、項目オブジェクト等の技術により再利用部品の構築を可能にした。

2002 年には、これらの部品を使って開発したプログラムのメインルーチンが汎用化できる点に着目し、データ構造にあわせた登録、照会、変更、削除等のコンポーネント開発に着手した。理論的に考えられるコンポーネントは 200 種類以上あるが、コンポーネント内部の部品が再利用できる構造を考案することで、６ヶ月で 170 以上のコンポーネントを開発することができた。

コンポーネントの再利用を推進するためには、コンポーネントの利用者、すなわち各プロジェクトの SE、プログラマがコンポーネントを理解している必要がある。コンポーネントの理解が足りなければ、不適切なコンポーネントを選択し、かえって工数が増加する事態も考えられる。そこで、社内および協力会社の SE、プログラマに対してコンポーネントの説明会を開催し、コンポーネントの機能説明とともにコンポーネントのソースコードの説明を行った。コンポーネントを利用した開発では外部設計が重要となる。外部仕様書の仕様がコンポーネントの仕様に合っていなければプログラマは異なる部分をコーディングしなければならない。外部設計の精度を上げるために、外部設計を担当するＳＥにコンポーネントの演習を中心とした教育を行い、理解度を高めた。更に外部仕様書のレビューを行い、コンポーネントが正しく利用できる画面設計になっているかチェックすることで、プログラマがスムーズに開発できるようにした。

次にメーリングリストを立ち上げ、プログラマに対する支援を行うと同時に社内各プロジェクトからのニーズを吸い上げ、コンポーネントの機能強化も継続的に行える体制を築いた。

以上の取り組みにより再利用可能なコンポーネントの開発と各プロジェクトでの再利用が全社的に行える体制が構築できた。パイロットシステムでも再利用の効果が出ており、今後は更にコンポーネント化の範囲を広げていきたい。

目　次

1. はじめに

近年、ソフトウエアの部品化がやりやすいオブジェクト指向言語 Java の登場により、コンポーネントを利用したシステム開発が注目されている。従来の労働集約型の開発から再利用可能なコンポーネントを組み合わせる組立型の開発形態に移行することで飛躍的な生産性の向上、コスト削減、納期短縮、品質向上を目指している。しかし、コンポーネント流通の実態を見るとコンポーネントの再利用は思うように進んでいない。一般的には再利用可能な範囲はビジネスロジック部分（モデル）とされているが、我々は過去の経験から事業部独自のロジックが入るモデル部は再利用が難しいと判断した。また、オブジェクト指向設計を採用すれば、データとプロセスのカプセル化により、多くのプログラムにシステム固有の項目名称等が埋め込まれるため再利用が難しいと判断した。

当社では事業部が業務で使用しているコード体系からデータ構造を分析するデータ中心設計を採用し、開発生産性や保守性の向上を行ってきた。コンポーネントの再利用においても事業部に最適なデータ構造の上で利用できるコンポーネントの開発を目標とし、1999 年に Java フレームワークを開発した。この際、当社独自の技術である項目オブジェクトを考案し、画面やＤＢの入出力を汎用化し、再利用可能な部品を作成することに成功した。

2002 年には、これらの部品を使って開発したプログラムのメインルーチンがコンポーネント化できる点に着目し、データ構造にあわせた登録、照会、変更、削除等のコンポーネント開発に着手した。本論文ではメインルーチンのコンポーネント化とコンポーネントの再利用について報告する。

2. 取り組みの背景と狙い

2.1. 取り組みの背景と狙い

当社では、1999 年にソフトウエアのコンポーネント化を目標とし、画面やＤＢの入出力等、再利用率が高いと思われるシステムよりのものから部品化を始めた。約３年間、全社的にフレームワークを利用し、フレームワークの機能強化を続けた結果、フレームワークの完成度が高まり、さらなる生産性向上のためには再利用ができていないメインルーチン（画面出力および画面遷移制御）またはビジネスロジックのコンポーネント化が不可欠であった。前者は再利用率が高く、全社的に効果が得られると判断したため、2002 年からメインルーチンのコンポーネント化に取り組むことにした。

今回の取り組みの目標は、以下の２つである。

(1) コンポーネントの再利用によりプログラマのコーディング量を減らし、生産性を上げること。
(2) コンポーネントの開発と再利用が継続的に行える体制を整備すること。

2.2. コンポーネント化の前提条件

画面出力および画面遷移のコンポーネント化にあたり、次の２つを前提とした。

(1) データ中心設計の適用

当社では、データ設計手法として佐藤正美氏の考案したＴ字形ＥＲ手法を利用している。完成したＥＲ図は正規化された状態でそのままリレーショナルデータベースに実装するものとする。正規化することで、データの更新ロジックを単純化でき、コードの一般化が容易になる。

(2) 項目オブジェクトの利用

プログラムの一般化を行うためにはプログラムに可能な限り固有名詞を含めないことが重要であると考えている。当社では、項目名等の固有名詞をパラメータ化するために項目オブジェクトを使用している。今回の取り組みの範囲ではないが、前提となるので参考までに説明しておく。

項目オブジェクトは、DB のテーブルの項目に対応したオブジェクトであり、項目名やデータ型、桁数、必須指定といった属性とエラーチェック等の処理をカプセル化したものである。項目オブジェクトを使用した画面表示の動作を図 1に示す。プログラマの作成するメインプログラム中には画面に表示したい項目名がパラメータとして記述されている。データ入力画面を表示するプログラムでは明細入力表示部品が項目オブジェクトにアクセスし、項目名称や桁数等の情報を基に HTML 形式で画面を出力する。データの登録プログラムでは、入力チェック部品が項目オブジェクトにアクセスし、桁数やデータ型、必須等のチェックを行い、エラーがなければデータをＤＢに登録する。項目オブジェクトは Java で記述されたオブジェクトであるため、パスワードは６文字以上といった制限も簡単に実装することが可能である。項目オブジェクトの考案によりパラメータを与えるだけで画面表示を可能にした。

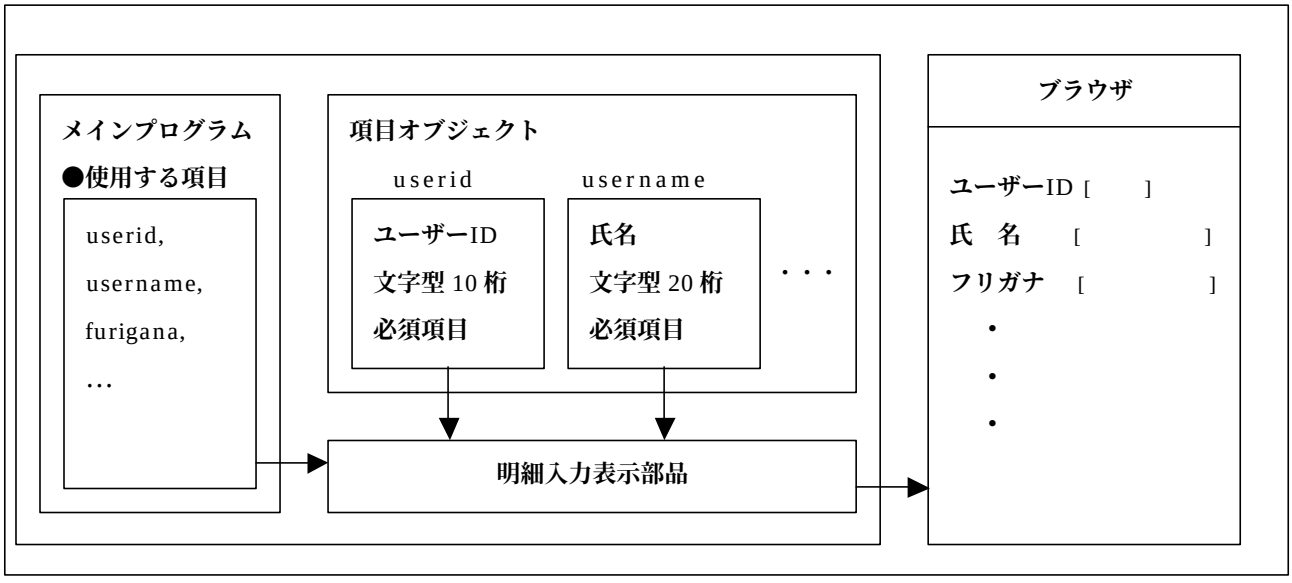

図 1. 項目オブジェクトを使用した画面表示

3. 画面および画面遷移のコンポーネント化

3.1. 業務プログラムの分析

当社では、1997 年よりイントラネットによる基幹システムの構築を行っており、既に数千本のプログラムがイントラネット上で稼働している。メインルーチンには、画面表示部品を組み合わせて画面を出力する部分と画面遷移を制御する部分で構成される。画面出力および画面遷移を部品化するためには、何らかの基準でこれらのプログラムを分類しなければならない。ブラウザを利用したイントラネットのシステムでは、ブラウザとサーバーのセッションが毎回切断されるため、クライアント＆サーバーシステムに比べ画面設計の制約が多い。そのため、似通った画面構成や画面遷移をもったプログラムが多くなる。例えば、照会プログラムは図 2の(a)に示すように検索条件を入力し、該当データを一覧表示し、一覧の中から照会したいデータをクリックすると明細が表示されるという画面遷移になることが多い。

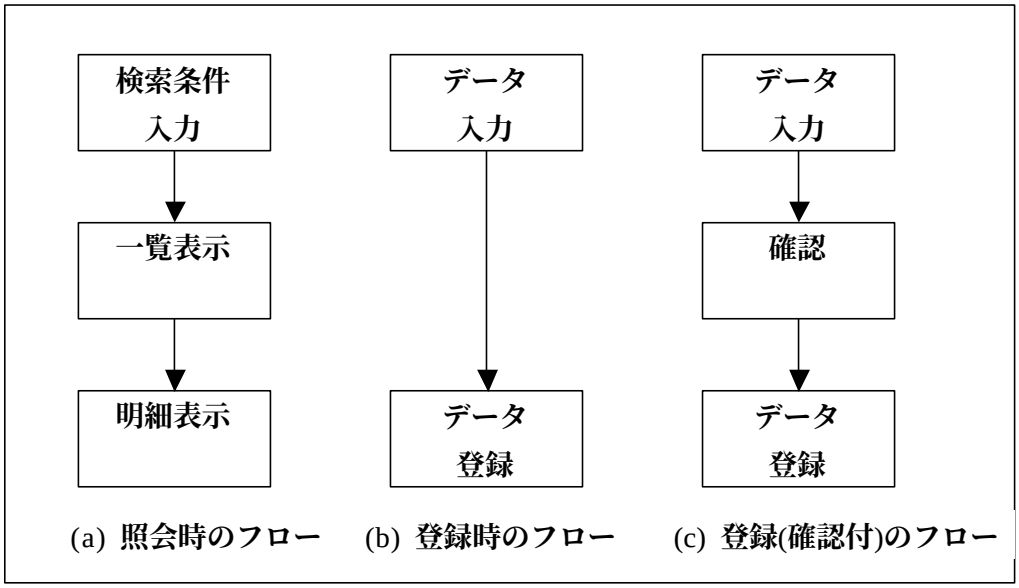

図 2. 画面遷移の例

我々は過去に作成したプログラムを分析し、下記の４つの観点でプログラムを分類できると考えた。

(1) 機能

基幹システムの多くのプログラムは、照会、登録、変更、削除の４つの機能を持っている。しかし、照会のみのプログラムや登録、変更が可能で削除機能を持たないプログラムも存在したため、これらの機能を別々のコンポーネントで実装し、プログラムは複数のコンポーネントを組み合わせて実装することにした。本論文では、照会や登録といった単機能を持つコンポーネントをパターンと呼ぶことにする。基幹システムで必要な機能はこれら４つ以外にも帳票出力や CSV ファイルの出力、コード一覧用のサブウインドウ等が考えられる。

(2) 画面構成

パターンは更に画面構成によって分類することができる。代表的に画面構成は、１つのテーブルを扱うものと受注データのように基本情報１件と複数の明細データを持つヘッダ・ディテールの構造である。画面構成はデータ構造によって決まることが多い。

(3) データの種類（リソース、イベント）

当社が採用している「Ｔ字形ＥＲ手法」はエンティティを時系列の並びに意味をもつイベントとそれ以外のリソースに分類することが１つの特徴である。リソースは、社員や製品、顧客といったエンティティであり、一般的にはマスターメンテナンスプログラムでデータのメンテナンスを行う。マスターメンテナンスプログラムはビジネスロジックを持っていないことが多く、最も再利用しやすいプログラムである。一方、イベントは、受注、出荷、請求といったタイムスタンプを持つエンティティである。一般的にイベントを扱うプログラムは個別に設計して作成することが多い。イベントとリソースの違いを分析した結果、以下の違いがあることがわかった。

(a) イベントのテーブルにはリソースのコードが含まれており、顧客名等のリソースの名称を表示するためには、リソースを JOIN しなければならない。
(b) 後続イベントが発生すると、先行イベントの変更、削除にロックがかかる。例えば、出荷済みの受注データは勝手に受注数量を減らしたり、受注を取り消したりすることはできない。これらの操作を行う場合、ロジックが必要となる。
(c) 未処理、未検収といった状態を管理しなければならない。

これらの機能を一般化して実装することで、従来部品化が不可能と思われていたイベントを扱うプログラムも再利用可能なコンポーネントとして実装することを可能にした。

(4) 画面遷移

同じ機能を提供するプログラムでも画面遷移にバリエーションを持つことがある。図 2の(b)と(c)は登録プログラムのバリエーションを示す。(c) はデータ入力後に確認画面を表示するため、画面数が多い。

これらの観点から既存のプログラムを分類し、これらの組み合わせ（機能×画面構成×データ種別×画面遷移）を考えれば数千本のプログラムが約 200 種類に分類できることがわかった。

3.2. サブメニューによるパターンの連携

抽出されたパターンの機能を更に分析すると新規登録以外のプログラムでは、データを特定する機能とデータを操作する機能が含まれていることがわかった。例えば、データを変更する場合、検索条件を入力後、一覧画面を表示し、変更したいデータをクリックして変更画面を呼び出す。検索

条件入力から一覧表示までの画面は、削除機能でも同様に必要となる。これらの画面は、パターン毎に個別に独立して実装することも可能であるが、エンドユーザーの操作性を考慮し、図 3に示すように照会機能を使ってデータを１件表示させ、その画面から変更や削除画面に移動する方法にした。この方法では、組み合わされたパターンがお互いに連携し、サブメニューを表示する仕組みが必要となる。例えば、データが一覧表示されている時には、CSV ダウンロードのサブメニューが表示され、データが一件表示されている場合には更新、削除のサブメニューが表示される。我々はデータを一件表示する場合と複数件のデータが表示されている場合で異なる画面区分を設定し、画面区分に応じたサブメニューを表示できるようにすることで、大量のパターンの連携が容易にできるようにした。

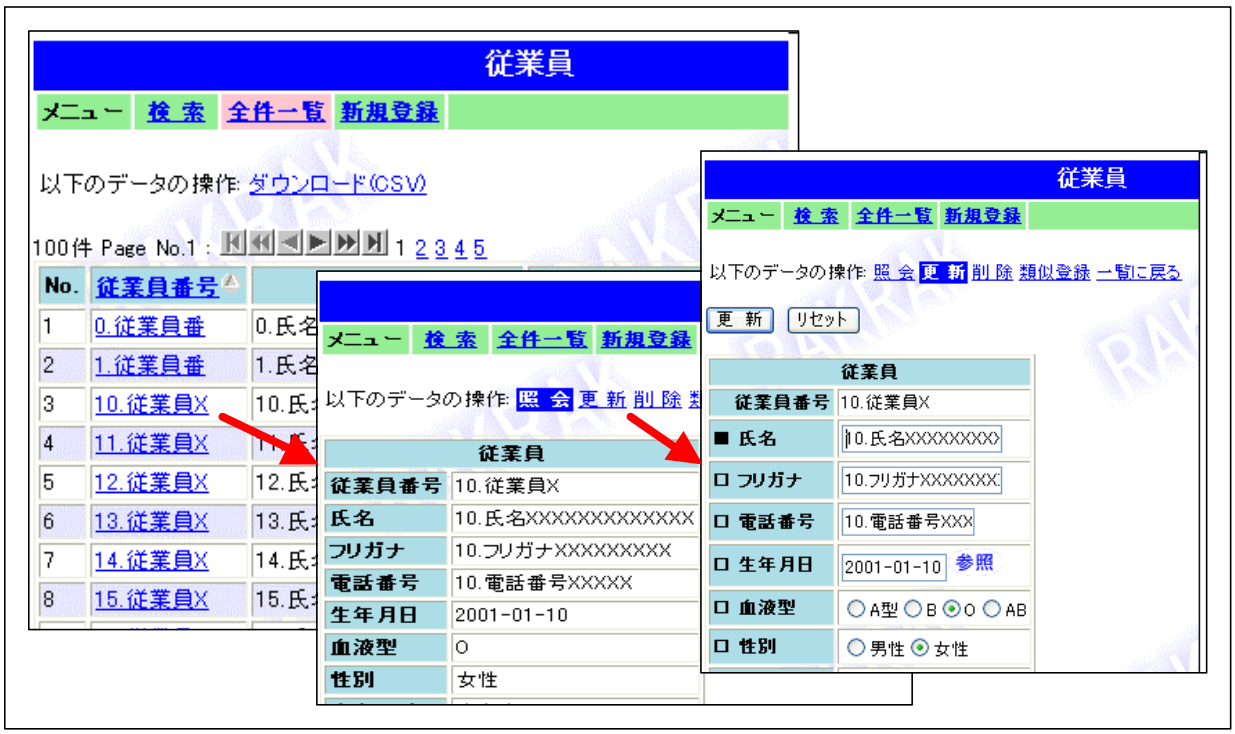

図 3. 操作性を考えた画面遷移の例

3.3. パターンの実装

過去のプログラムを分析した結果、約 200 種類のパターンが必要であることがわかった。これらのパターンを短期間で開発し、社内プロジェクトにリリースする必要がある。この際、各プロジェクトからの要望に応じて機能拡張することが予想されるため、保守性も十分高めておく必要がある。

そこで、200 種類のパターン内部のコードを部品化し、再利用できないか検討した。全体のコード量を抑えることで保守性を高め、再利用率を高くすることで品質を高めたい。パターンに実装する機能は、画面出力と画面遷移であるため、これらを別々に実装することにした。ここでは前者をスクリーン部品、後者を画面遷移制御部品と呼ぶことにする。

図 4にパターンの内部構造を示す。図の右側にあるスクリーン部品は、検索条件の入力やデータ

登録のためのデータ入力画面を表示するための部品である。スクリーン部品を開発すれば図 2の(b), (c) の画面でデータ入力とデータ登録の画面が再利用可能となり、(b)のパターンが完成すれば、(c)のパターンは確認画面を表示するスクリーン部品を追加するだけでよい。更にここで作成した確認画面は、更新機能の確認画面でも利用できるようになる。

しかし、スクリーン部品の組み合わせで新しいパターンを作るにはもう１つ課題があった。データ入力画面で入力したデータは、次に起動されるデータ登録プログラムでデータをチェックし、ＤＢへの登録を行う。データの入力部とデータのチェック部が別プログラムになれば、スクリーン部品間に依存関係が生まれ、スクリーン部品の組み合わせが制限されてしまう。そこで我々はスクリーン部品を３つの機能に分解することで、スクリーン部品の組み合わせの自由度を高めた。

図 4でパターンの動きを説明する。①でブラウザに入力画面が表示される。データを入力し、登録ボタンを押すと②に示すように１つ目のスクリーン部品のエラーチェック機能が呼び出される。エラーがなければ③データ登録用のスクリーン部品が呼び出され、エラーチェックせずにデータをDB に登録する。このようにエラーチェック部分を登録用スクリーンから分離したことで、１つ前の画面が入力画面であっても確認画面であっても組み合わせ可能な部品を実装できるようにした。

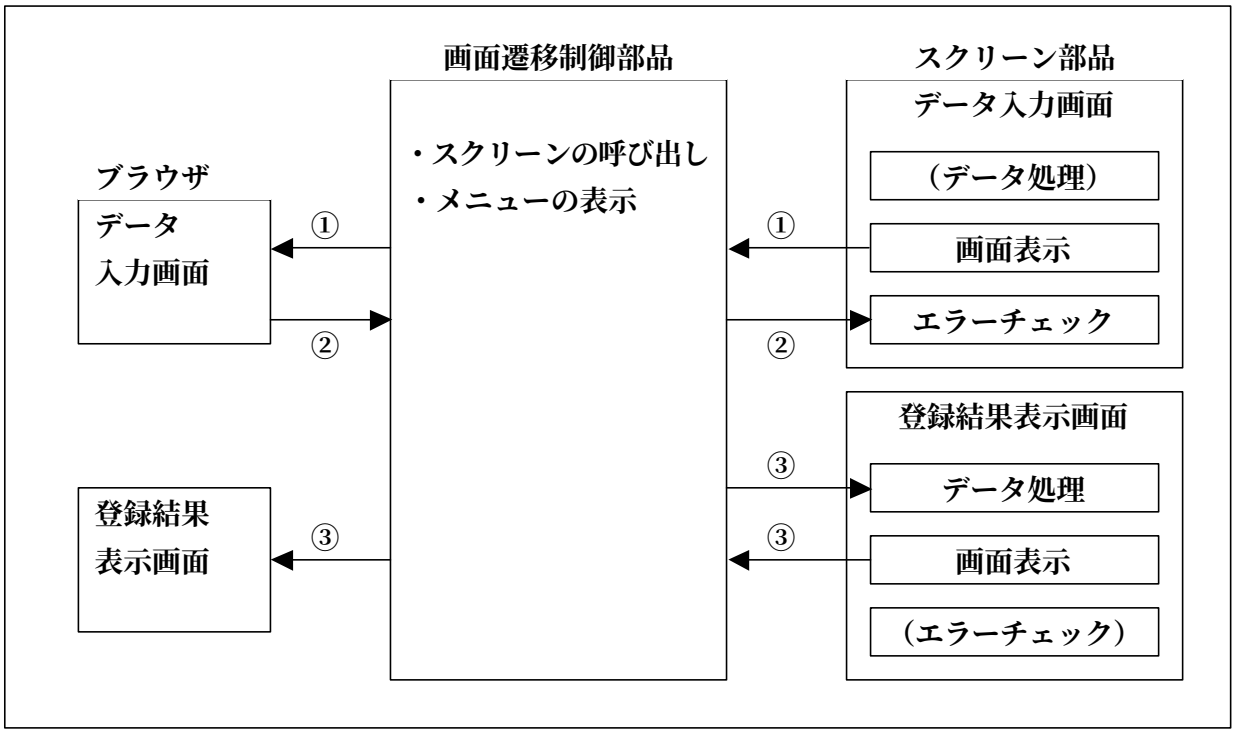

図 4. パターンの内部構造

スクリーン部品によりパターン内のコードの再利用を実現したが、この構造であれば画面構成毎にスクリーン部品を作成する必要がある。基幹システムで扱うデータ構造は多岐にわたるため、個別に開発すれば膨大な数のスクリーン部品を開発する必要がある。そこでスクリーンを画面の構成

要素毎に分割することにした。図 5はヘッダ・ディテール構造の画面を表示する際の構成を表示している。ヘッダ部を表示するスクリーン部品は、単一テーブルのデータを明細入力するスクリーン部品と同一でものであるため、ディテール部のスクリーンを追加開発するだけでヘッダ・ディテールのパターンが開発できる。

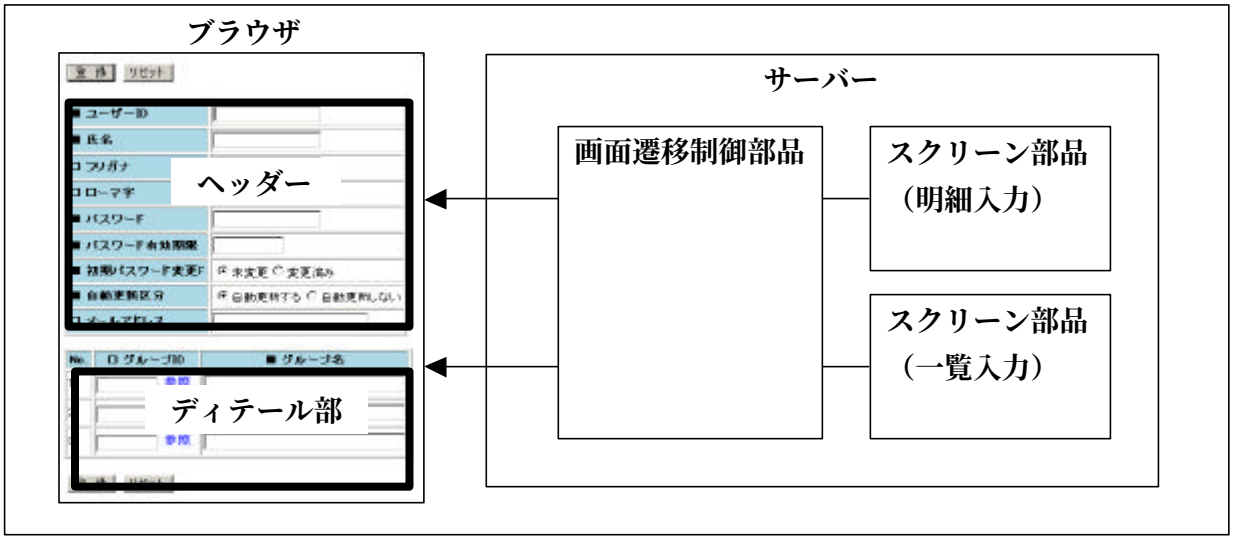

図 5. スクリーン部品の分割

このようにパターンの実装を工夫することで、効率的にパターンが開発できるようになり、表 1に示す１７６個のパターンを約６ヶ月で作成することができた。

表 1. 作成したパターン

機　能	リソース用	イベント用
照会	9	16
登録	17	30
変更	16	25
削除	13	25
帳票		3
サブウインドウ	8	
その他（CSV, グラフ等）	7	7
合　計	70	106

3.4. 利用方法の検討

(1) パターンの選択方法

豊富なパターンを利用して生産性を上げるためには、外部設計時に 170 以上あるパターンの中から最適なパターンを選択することが重要である。幸い、開発したパターンは４つの観点で分類されているのでネーミングルールを工夫するだけで簡単にパターンが選択できるようになった。表 2 にパターンのネーミングルールを示す。例えば、EntEFa はイベント用のヘッダ・ディテール用の登録画面である。最後の Fa は F が明細入力、a が一覧入力を示し、画面の上部に明細が、下部に一覧が表示されることを示している。通常、データ構造は画面構造に反映されるため、ER 図が確定した段階である程度利用できるパターンが絞られてくる。このようにネーミングルールにより最適なパターンが簡単に選択できるようにした。

表 2. パターンのネーミングルール

桁数	内容
１～３	処理内容 Inq:照会, Ent:登録, Upd:変更, Del:削除, Rpt:帳票等
４	データの種類（R:Resouce, E:Event）
５～	画面構成（5 文字目は大文字、６文字目以降は小文字） D:明細表示、L:一覧表示、F:明細入力、A:一覧入力
	画面遷移（オプション。大文字） C: 確認付

(2) パラメータの設定

パターンには、画面に表示する項目や更新するテーブル等の情報が一切含まれていないため、これらの情報を何らかの方法で定義する必要がある。プログラムの開発効率を上げるため、これらのパラメータはプログラム中に記述するのではなく、XML(eXtensible Markup Language)を利用して記述することにした。この XML ファイルを XPD(XML Program Definition)と名付けた。XML を利用することでコンパイル無しでパターンが利用できるため、テキストエディタのみでプロトタイプの開発も可能となった。

図 6に従業員データのテーブル登録を行うＸＰＤのサンプルを示す。<Pattern>は利用するパターンを指定するタグで、照会、登録、変更、削除の４つのパターンが指定されている。その他、更新するテーブル、検索キー、一覧表示項目、明細画面の表示項目がそれぞれ、<Table>、<KeyFields>、<ListFields>、<PrintFields>で指定されている。このようなパラメータファイルを作成するだけでコンポーネントが簡単に利用できる。

```
<Program NAME="prog0100">
      <Title>従業員</Title>
     <Title  Lang="1">emp</Title>
      <Pattern>PtnInqRD</Pattern>
      <Pattern>PtnEntRF</Pattern>
      <Pattern>PtnUpdRF</Pattern>
      <Pattern>PtnDelRD</Pattern>
      <PP>
            <Table>emp</Table>
            <KeyFields>emp_no</KeyFields>
            <ListFields>emp_no, name, furigana, tel</ListFields>
            <PrintFields> emp_no, name, furigana, tel, birthday ...</PrintFields>
            <SQL>
               <Select>emp_no, name, furigana, tel, birthday ...</Select>
                 <From>emp</From>
                 <Where></Where>
                 <Order>1</Order>
            </SQL>
```

図 6. パターンに与えるパラメータの例

3.5. ビジネスロジック呼び出し（プラグイン）

図 7にシステム全体のアーキテクチャを示す。ブラウザでサーバーにアクセスすると要求を受け取る Servlet が起動される。Servlet は、パラメータを記述したＸＰＤを読みとり、パターンを呼び出す。パターンは XPD にしたがって画面出力と DB の入出力を行う。更にプログラムにロジックを埋め込む場合、パターン用プラグインを開発する。例えば、出荷入力を行った場合、入力データがパターンによって出荷テーブルに書き込まれる。この際、在庫の引き落としを行う必要があれば、在庫管理のロジックを実装し、プラグインでそのロジックを呼び出せばよい。プラグインには、ＤＢアクセスの前後や画面出力の直前等に呼び出される約 50 種類のメソッドがあり、任意のタイミングでビジネスロジックを呼び出すことができる。

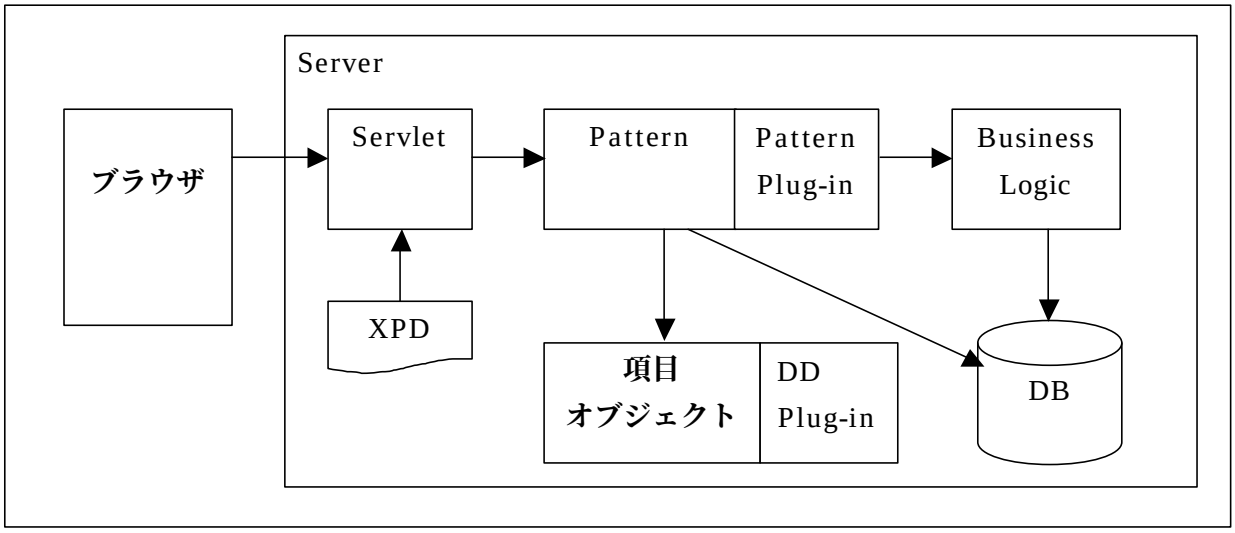

図 7. アーキテクチャ

4. コンポーネント再利用の推進

4.1. コンポーネント再利用の推進体制

コンポーネントの再利用を推進するために以下の３つの課題に取り組むことにした。

(1) 各プロジェクトで必要となった機能を迅速にコンポーネントに追加すること（有用性の向上）

(2) 各プロジェクトのＳＥ、プログラマが簡単にコンポーネントを利用できること（可用性の向上）

(3) コンポーネントの品質が十分高いこと。

図 8に示すコンポーネント再利用の推進体制を示す。上記３つの課題をクリアするために３つのチームを編成した。

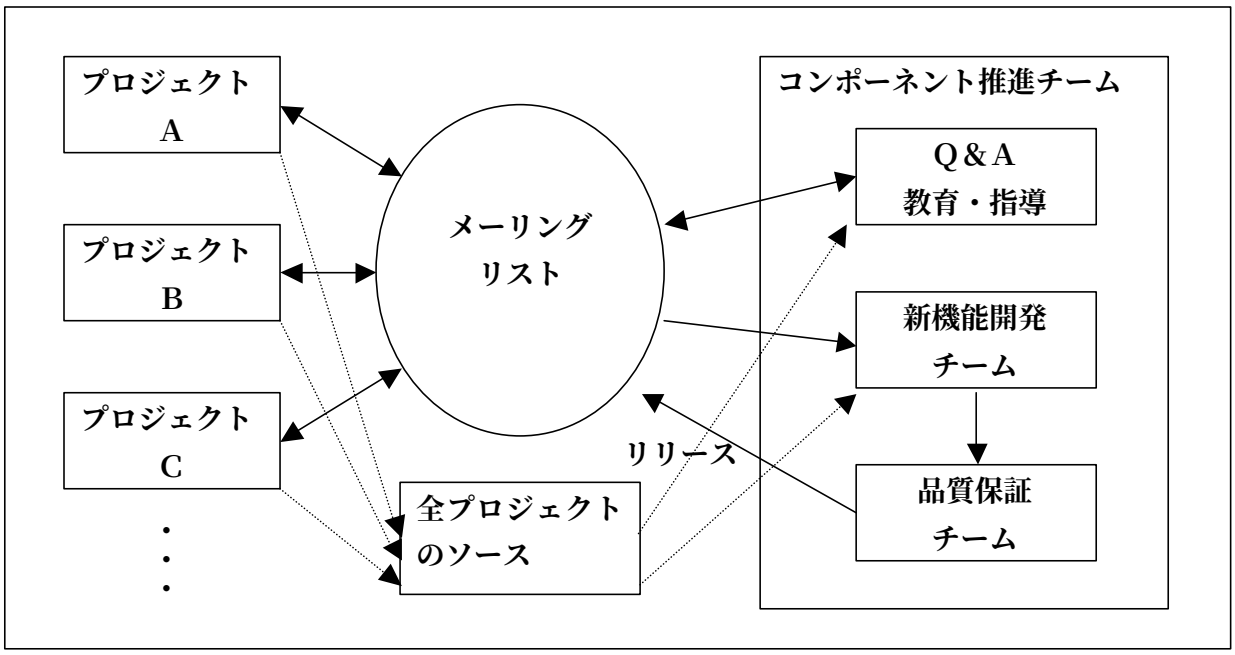

図 8. コンポーネント再利用の推進体制

4.2. 有用性向上の取り組み

コンポーネントの再利用を進めるためには、各プロジェクトからのニーズを汲み上げ、継続的に新しい機能を迅速にリリースする必要がある。そこで、新機能開発チームを設置し、メーリングリストで寄せられる要望の中で再利用性が高いものに関しては１週間以内で開発できるようにした。

新機能は既存のコンポーネントを変更するか、新コンポーネントを開発することで実現される。コンポーネントの変更を行う場合、既存プログラムに不具合が発生しないように慎重に仕様を検討する必要がある。しかし、コンポーネントの提供者の意図しない方法でコンポーネントが再利用されている場合があり、厳密に影響を調べられるように１日２回の頻度で各プロジェクトのソースを収集し、ソースを分析することで影響がないことを確認できるようにした。

4.3. 品質保証

日々拡張されていくコンポーネントの品質を保証するため品質保証チームを設置した。このチームは、毎週水曜日に開発チームから受け取ったコンポーネントを毎週金曜日にリリースする。２日間で毎週機能拡張されるコンポーネントの品質保証するために市販の自動テストツールを活用した。品質保証チームは、新機能を検証するシナリオを作成し、テストツールで動作を検証する。また、過去に作成したシナリオを使って互換性のテストを行い、合格すれば各プロジェクトに新しいコンポーネントをリリースする。このようにして品質の高いコンポーネントを毎週リリースできる体制にした。

4.4. 可用性向上の取り組み

これまでの取り組みにより再利用を妨げる技術的な問題はほぼ解決できた。最後に解決すべき問題は、各プロジェクトのＳＥ、プログラマの意識改革である。再利用を妨げる大きな要因の１つが「人の作成したプログラムを使いたくない(Not Invented Here)」、「使い方を覚えるより自分で開発した方が楽しい」といった心理的なものである。この問題は教育とＱ＆Ａサポートの２つの方法で解決することにした。

教育は、社内および協力会社を対象に３回にわたって約 120 名のＳＥ、プログラマにコンポーネントの教育を行った。コンポーネントを利用したシステム開発を行うためにはコンポーネントの仕様を理解すると同時に、コンポーネントの内部構造を知り、カスタマイズの難易度と工数見積もりができなければならない。そこでコンポーネントの教育では、コンポーネントの仕様と同時にコンポーネントのソースを解説することでＳＥの理解度を上げた。

しかし、その後の外部仕様書のレビューでＳＥのコンポーネントの理解度が十分でないことが判明した。外部仕様書のレベルでコンポーネントの再利用が難しい仕様になっていれば、プログラム開発のフェーズでも生産性の向上が期待できない。そこで、外部設計を担当するＳＥに演習を中心とするコンポーネントの講習を再度実施し、レベルアップを図っている。

4.5. コンポーネント開発者の養成

パターンが 100%利用できる場合、プログラマはビジネスロジックのみをプログラミングすることになる。しかし、実際のプロジェクトでは、パターンの適用率は 80%程度である。既存パターンで実現できないプログラムは各プロジェクトで独自に開発することになるが、ここで作成するプログラムも汎用的なパターンとして開発し、プロジェクト内や他プロジェクトへの展開を図るのが望ましい。そこで、各プロジェクトのプログラマに汎用的なパターンの作成方法を教育することにした。この教育はプログラマがその技術が必要になったとき自由に勉強できるように、ホームページ上に自習教材を提供する形式で実施した。プログラマは、演習問題と解答を見ながら汎用的なプログラムの作成方法を覚えていく。最後のページには認定問題があり、プログラマがその問題の回答を教育チームに送付し合格すればホームページ上に認定者として表彰することにした。現在、30 名以上のプログラマが認定されている。

5. 評価

図 9は参考文献２で引用されている ISBSG (International Software Benchmarking Standards Group)の業界参考データをグラフ化したものである。当社の平均的なシステムは 1000 ファンクションポイント（以下、ＦＰ）であるが、1125FP の規模のシステムでは、オブジェクト指向言語がメインフレームに比べ約 37%生産性が悪い。

図 10は 2000 年度および 2001 年度の生産性をＦＰ当たりの工数の相対値で示したものである。社内全プロジェクトを対象とし、内部設計から総合テストの工数を言語別に集計した。COBOL の開発はベテラン・プログラマが行っているのに対し、Java の開発は新入社員を含む初心者や経験の浅いプログラマが行っているが、部品の再利用の効果により 2001 年度は COBOL の約３倍の生産性が実現できている。

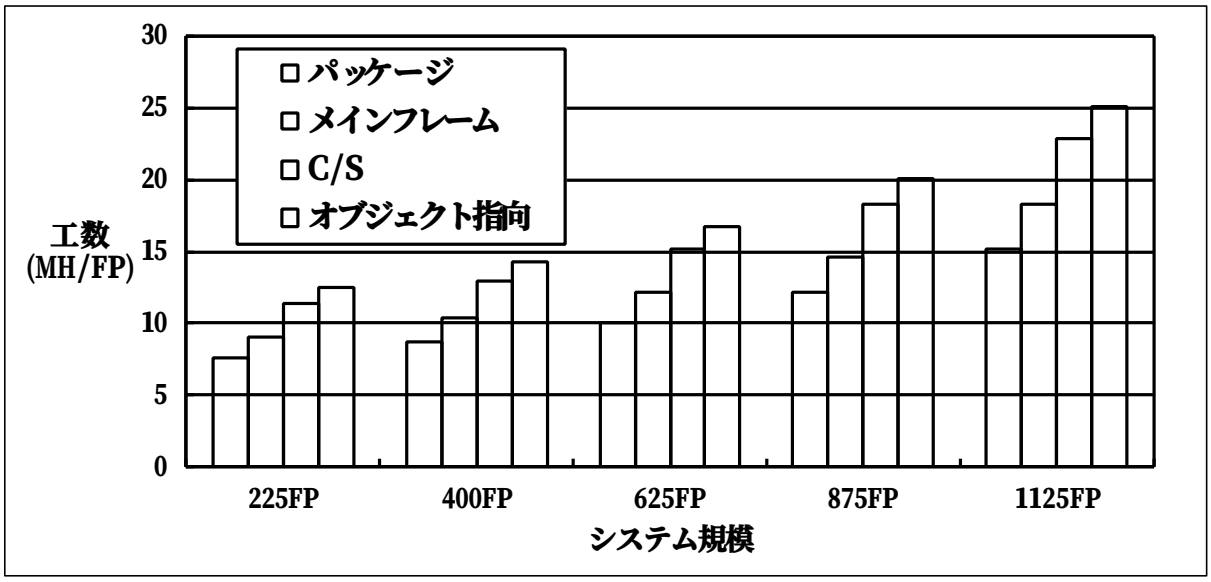

図 9. ISBSG (International Software Benchmarking Standards Group)の業界参考データ

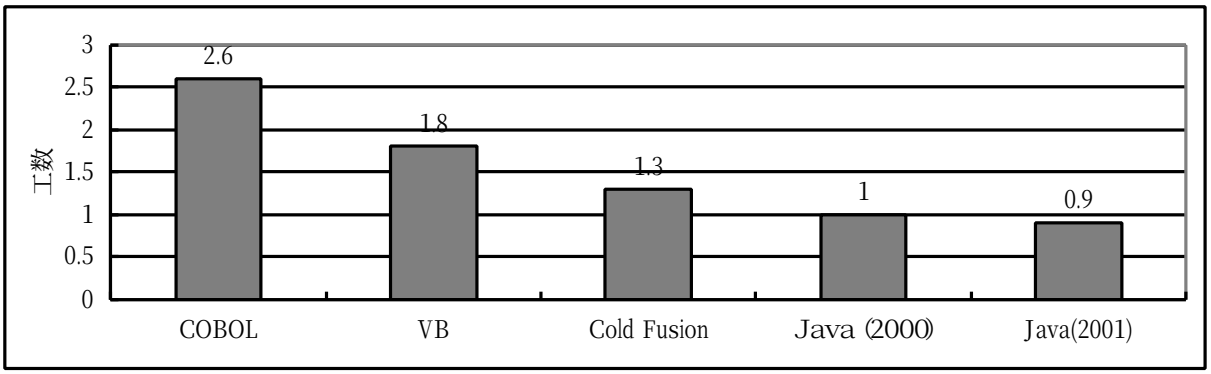

図 10. Framework を使用した Java の生産性（FP 当たりの工数の相対値）

今回の取り組みの効果を検証するために 660FP の出張旅費システムでコンポーネントの再利用を実施した。このシステムはビジネスロジックが多く、ワークフローを利用するため、技術的にも難しいものである。現在、総合テスト中で全体を通した生産性の数値は集計できていないが、内部設計～プログラミングのフェーズで 2001 年度の実績に対して約 10%の工数削減が実現できており、パターンの効果を確認することができた。標準的なシステムでは更に生産性の向上が期待できる。

また、再利用可能なコンポーネントが継続的に蓄積され、再利用される体制が構築できたことも大きな成果だと考えている。

6. 最後に

ソフトウエアの再利用については昔から議論されており、ソフトウエア開発の生産性を大幅に向上させる非常に重要な技術だと考えられる。Java 言語の普及により部品化のための実装技術は確実に進歩しているにもかかわらず、再利用の取り組みがあまり報告されていないのは残念である。

我々は、「再利用可能なコンポーネントは、項目名等の各プロジェクト固有の固有名詞を埋め込んではならない」という点に注意しながら再利用範囲の拡大を行ってきた。今回の取り組みによりビジネスロジック以外の部分の再利用に関しては技術的に問題ないと考えている。残るビジネスロジックの部分についても販売管理や生産管理といった分野ではある程度コンポーネント化できる機能が見えてきている。これまでは、社内でビジネスロジックを再利用するニーズは少なかったが、分社化の流れにより子会社が独自に経理システム、販売システム、生産管理システム等を短期間で開発する必要があり、ビジネスロジックを含むシステム全体の再利用のニーズが高まっている。システムを手作りする時代から、組み立てる時代へ。システム開発の工業化へのチャレンジを続けていきたい。

今回の報告がシステム開発の効率化を検討されている方々にとって何らかの参考になり、またソフトウエアの再利用について検討するきっかけになれば幸いである。

参考文献

１．著者名：中村伸裕・谷本收、題名：第３８回ＩＢＭユーザー・シンポジューム論文集「オブジェクト指向技術による Java フレームワークの構築」、発行年：2000 年 5 月、ページ数：１ページ
２．著者名：佐藤正美、題名「T字形ＥＲデータベース設計技法」、出版社名：ソフトリサーチセンター、発行年：1998 年 10 月 25 日
３．著者名：デービッド・ガーマス著、題名「ファンクションポイントの計測と分析」、出版社：ピアソン・エデュケーション、発行年：2002 年 12 月 15 日、ページ数：５１ページ